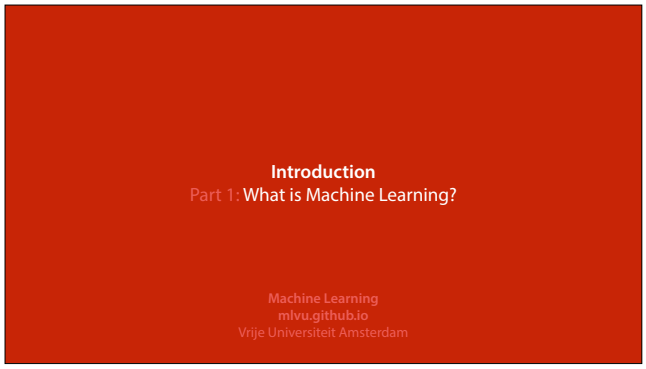


In this first lecture, we we look at what machine learning is, and at some of the basic definitions. We will also have a quick look at some simple methods that you might use to do machine learning, although we will save most of the details of these methods for later.

We'll start with a simple example.

In the 1990s, the US postal service processed billions of letters each year. Many of them had hand-written addresses, like this one. To automate at least part of the process, it would be very helpful if computers could read, if not the whole address, at least the zip code.

Reading digits is pretty easy for us, almost all humans can do it. The problem is that this is one of those tasks that humans know how to do without knowing how it is that they do it. We can all read these digits, but if we met somebody who couldn't, none of us could tell them exactly what steps they should follow to do what we do. We might say something like "a two is always a continuous line, with no loops, with a curve at the top a corner in the bottom left and a flat line at the bottom". But that doesn't tell how we recognize a line, a corner and a curve in the first place. It also doesn't explain why we recognize the second digit in this zip code as a two, even though it violates this rule.

In short, even if we have some idea of what we're doing, we can't make the process precise enough to turn it into an algorithm. So how did we acquire the ability to recognize digits? We certainly weren't born with it.

image source: **https://rafalab.github.io/dsbook/introduction-to-machine-learning.html**

We **learned**, of course. We were shown examples of different digits and somebody told us which was which, and after a while, we figured out the general idea: what makes a 3 a 3, despite the many different ways of writing it.

Machine learning is the practice of applying the same idea to computers. Instead of providing a set of instructions to follow step by step, like we normally do, we provide a large number of ***examples*** of the sort of thing we want the computer to learn, and then we try to figure out a program that recognizes the regularities in the examples and ignores the details.

image source: **https://www.pbslearningmedia.org/**

resource/sesame-number-of-the-day-0/song-number-of-the-day-0-sesame-street/

Here is another problem that can be attacked with machine learning: playing chess. In this case, we don't *need* machine learning. We understand chess well enough that we can design a program that learns nothing; it simply follows instructions, but still is good enough to beat the best grandmaster. In this picture we see Deep Blue, a system that used no machine learning, playing (and defeating) the then world champion Garry Kasparov in 1997.

However, we do have many examples of how humans play chess, and we *could* have a computer learn from this. In fact, the current best chess playing computer program AlphaZero, uses a lot of machine learning (although strictly speaking, it doesn't learn from human games).

A more complex problem is that of designing a self driving car.

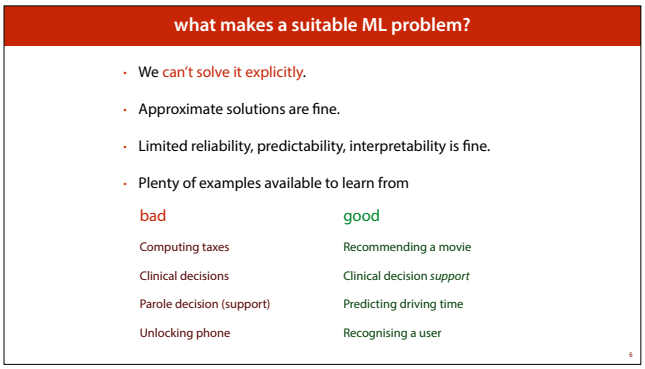
Again, we wouldn't be able to design a set of rules that can always be followed to drive perfectly. Many important aspects of driving: following the curve of the road, recognizing pedestrians and traffic signs, are simply too complex and too poorly defined to simply tell a computer what to do in a set of instructions to be followed like a recipe.

Many of these problems can be isolated. For instance, we can collect a dataset of views from the car window, and train a model to recognize whether a stop sign is present. That doesn't give us a self-driving car, but it

solves part of the problem. For basic road following, we can observe a human driver, and see what pressures they apply to the steering wheel to keep the car straight.

Of course, even if we successfully train all these separate modules, we need to make sure that they then work together when we integrate them into a large system. This is very much still an open problem.

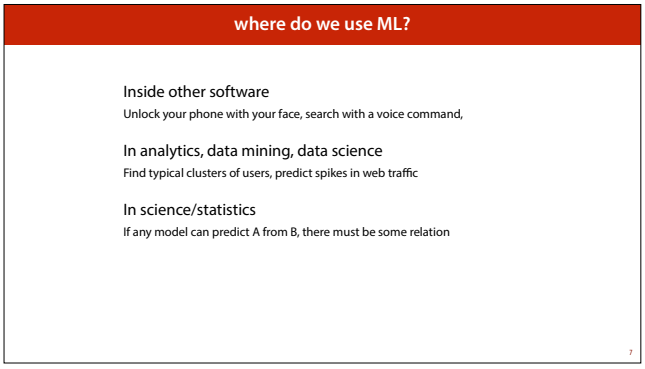
image source: the oatmeal, **http://theoatmeal.com/blog/google_self_driving_car**



In a chess game, approximate solutions are acceptable. In zip code recognition, it may cause the odd letter to be delayed, but that could be an acceptable sacrifice. In self-driving cars, this because a more complicated question, and we need to be really sure that we don't place too much trust in a pedestrian-recognizing module. (This is one of the reasons why self-driving cars are not living up to the hype we saw a few years ago).

We don't want to use ML for tax computation, because approximate solutions are not acceptable, and an explicitly solution is easy to calculate. Recommending a movie is a great use case, since we have to explicit solution, and approximate solutions are fine: we can simply suggest a lot of movies to the user and let them pick. This is usually the reason why approximate solutions are fine: we can embed the ML module in an interface that gives a user some control. In other words, we don't give the ML system full autonomy, it is used to make suggestions to a human user.

Fun fact, ML systems do exist for parole decisions, and face detection is used for unlocking many phones. These are very problematic developments.



a broad definition

(from expertsystem.com)

Machine learning [...] provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

8

Here is a decent definition of Machine Learning that covers most of he important aspects. It's a system (i.e. a computer running a program). It improves its behaviour based on experience, and the resulting behaviour has not been explicitly programmed.

This kind of definition suggests a system that learns and acts like a human being. It continuously updates its "mind" while also constantly making decisions and taking actions based on the information it has.

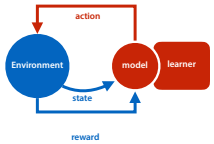
quote source: http://www.expertsystem.com/machine-learning-definition/

---

the broad view: an intelligent agent

reinforcement learning: taking actions in a world based on delayed feedback.

lecture 13



online learning: predicting and learning at the same time.

not in this course

9

This kind of systems is often called a learning *agent*. There are various subfields of machine learning that deal with such a broad view of machine learning.

In **reinforcement learning**, we study true learning agents. We need to define the agent, the environment, and a reward system. The agent must learn to explore the environment, while also taking actions to maximize its rewards. Its actions also change the environment, meaning that what it chooses to do may invalidate what it has learned so far. clearly, this is a very complicated problem.

In **online learning**, we simplify the problem a little bit. We are no longer taking actions, we are only predicting. That is for each input we need to predict the right output, but what we choose to predict doesn't affect what we will see in the future. We are still learning **online**: that is, every input we observe requires a prediction, but it also serves as an example to learn from in our future predictions.

Dealing with all these problems at once is very complicated. In most cases, we don't actually need an agent that learns as it acts. In those cases, we can simplify the problem of machine learning a lot.

---

offline learning

Separate learning, predicting and acting

- Take a fixed dataset of examples (aka instances)
- Train a model to learn from these examples
- Test the model to see if it works, by checking its predictions
- If the model works, put it into production
  i.e. use its predictions to take actions

most of the course

10

In **offline learning** you separate the acts of

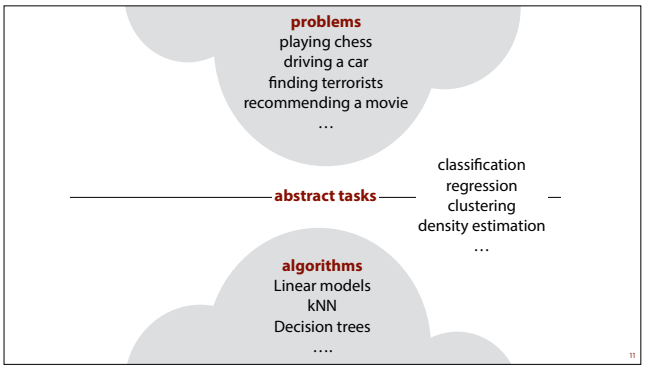
1. learning a model and
2. putting a learned model to use.

You gather a dataset of examples beforehand, you train a *model*, test it, and once you're sure it works well enough, you use that version of the model (for instance by sticking into a computer program). The finished program will never learn while it's running. There is no feedback loop between learning and running. Note that the acting part of the intelligent agent is eliminated entirely.

While this robs the exercise of some of its more exciting aspects (it's a far cry from building androids),

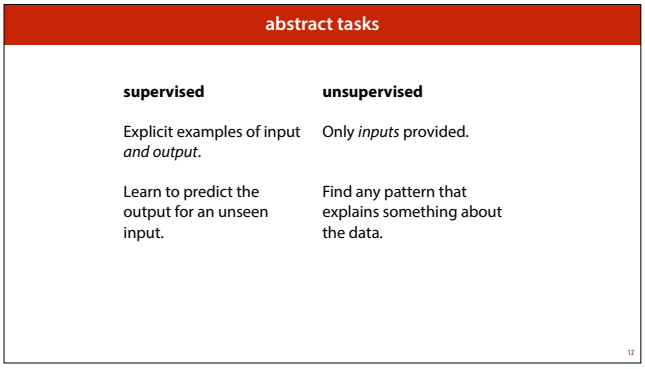
it still allows us to do something very useful: it allows us to learn programs that we have no idea how to write ourselves. For instance, we can learn a program that detects birds in pictures: we have no idea what kind of rules would be required, or how to design an algorithm to do it. Machine learning allows us to create such a program from a set of examples.

Almost all of this course will focus on offline learning.



Now, the main problem with machine learning is that **we want solutions that are applicable across domains**. You don't want to dedicate your entire life crafting a perfect self-learning computer chess program, and then find out that your ideas have no use for anything else. We want to solve the problem of machine learning **in general**: instead of studying each problem in isolation, we want solutions that can be applied to many problems.

To make this possible, machine learning is usually built on one of a few **abstract tasks** like *classification*, *regression* or *clustering*. If you have a practical problem, like chess playing, you find a way to abstract the problem of playing chess (or part of it) to the generic task of, say, classification, and then you pick one of many existing classification *methods*.



Abstract tasks come in two basic flavours: **supervised**, and **unsupervised.**

In supervised tasks, we have explicit examples of both inputs and the corresponding outputs. What we have to learn is the program that maps any input to the corresponding output. For instance, we may be provided with emails and given a label **spam** (advertising) or **ham** (genuine) for each. The task then, is to train a program to assign these labels to new emails.

In unsupervised tasks, there is no target value, only the data. All we can do then is to learn some structure in the data. For instance we can cluster students to see if

there are natural groups, or see if we can detect which financial transactions are “unusual”.

---

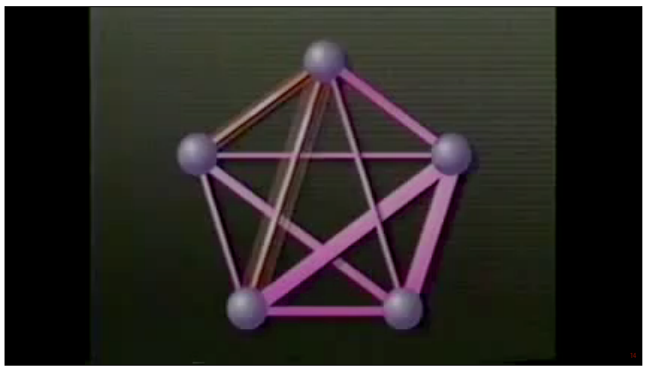
supervised learning tasks

Classification: assign a class to each example.

Regression: assign a number to each example.

13

These are the two main forms of **supervised offline learning**.



We’ll make this more precise in the next video, but to give you an idea, here is an example of classification being performed in the 1950s.

video source:

https://www.youtube.com/watch?v=7BtLqqJVP9w

https://archive.org/details/perceptron_documentary_excerpt

Machine Learning vs.:

Artificial Intelligence

Data Science

Data Mining

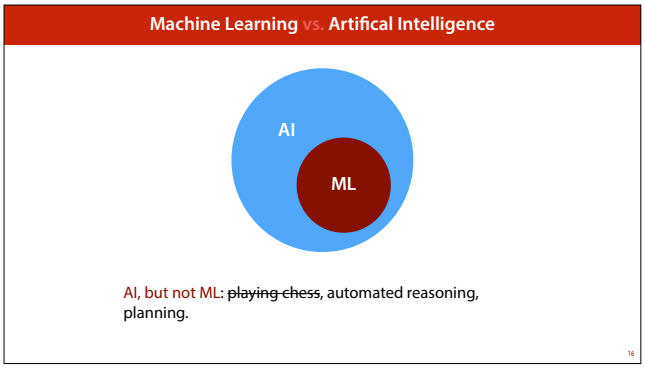
Information Retrieval

Statistics

Deep Learning

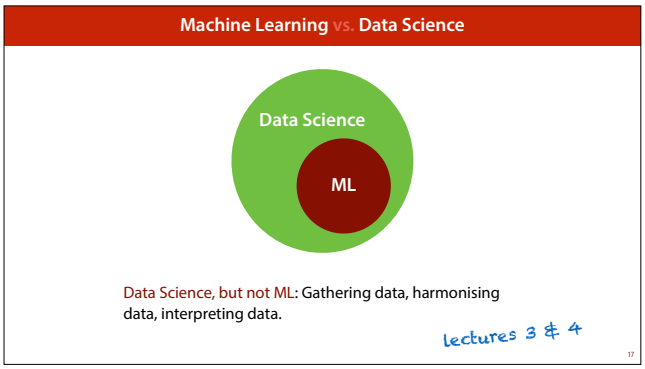
15

I’ve told you what machine learning is. Now let’s look at what it isn’t.To finish up, let’s see how ML relates to other fields of study.



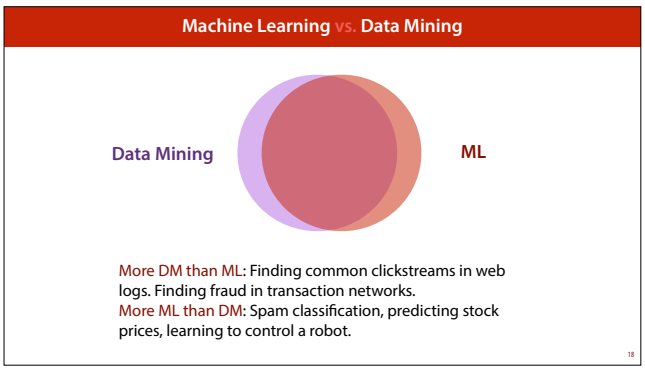
First up, Artificial Intelligence. Machine Learning is a subfield of AI. If we want to make a general AI that can do everything we can, it needs to be capable of learning. But there are many other problems and fields in AI that have nothing to do with learning.

Other subfields, like natural language processing, are greatly helped by ML techniques, but can also be tackled without.



Similarly all ML is Data Science, but not all Data Science is ML. Often, ML is used as part of a larger data science pipeline.

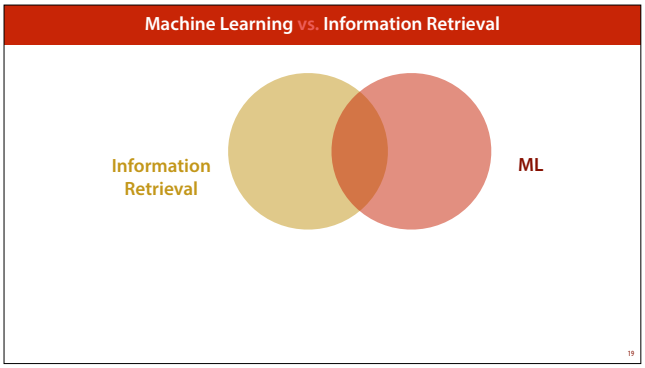
Thursday next week, we will look at some of the data science aspects required to perform Machine Learning experiments.



Here is a more subtle distinction: data **mining** and machine learning. Opinions differ, but I would characterise them as follows. Both analyze data, but they do so with different aims: machine learning aims to produce a *model* of the data, data mining aims to produce intelligence.

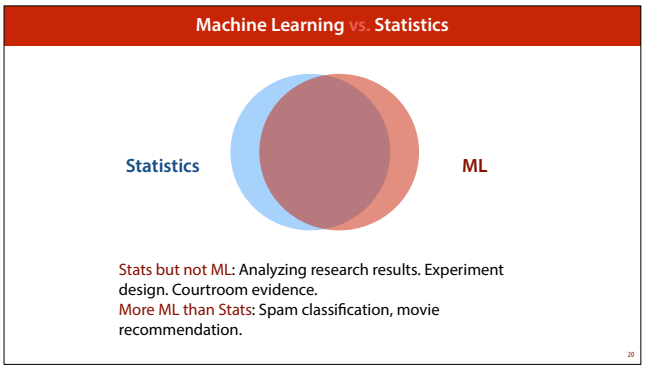
Another distinction is that machine learning focuses on prediction: trying to predict a target value for new data, whereas data mining tries to navigate and simplify the data so that it becomes useful for users.

If you have a dataset, but you expect never ever to see any new data from the same source, you can perform data mining on it, but performing machine learning on it is not much use (although your data mining will probably use machine learning *techniques*).



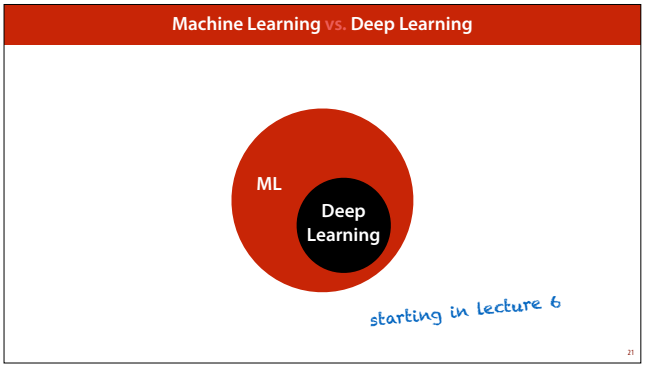
Information retrieval (building search engines) may look at first like a field that is completely distinct from ML. But on closer inspection, it turns out that you can model IR as a kind of classification task: your instances are documents, and your aim is to classify them into relevant or irrelevant (for a particular query).

This may seem a bit extreme, since the class imbalance is so high, but this has actually helped us in ML to think more clearly about problems with high class-imbalance (where ranking is a more appropriate way to think about the task than classification).



Here is another subtle distinction. Statistics and ML both focus on analysing data, and modelling it in some way. In fact, many of the models we use in ML were invented by statisticians before the name Machine Learning existed. The distinction isn't always clear, but the most important difference is that Statistics aim to get at the *truth*, whereas machine learning tries to come up with something that *works,* regardless of whether it's true.

Consider spam classification. We usually model emails as a bag of independently drawn words. This has nothing to do with the way emails are actually written. Still, it works well enough to let people control their in-box. Contrast this with proving in a courtroom that a particular piece of DNA evidence really puts a suspect at the scene of the crime. Here, we're interested in more than just getting a useful model that captures some of the data, we want the truth.

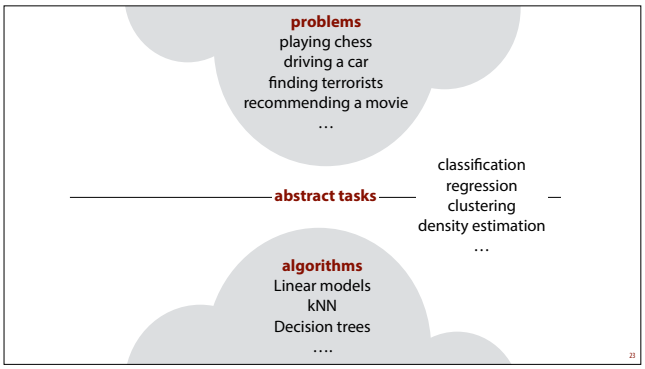


Finally, you may have heard a lot in the news in recent years about deep learning and wondered whether it is the same as machine learning or something different, or what. Here I can be clear. Deep Learning is a subfield of Machine Learning. All Deep Learning is Machine Learning but not all Machine Learning is Deep Learning.

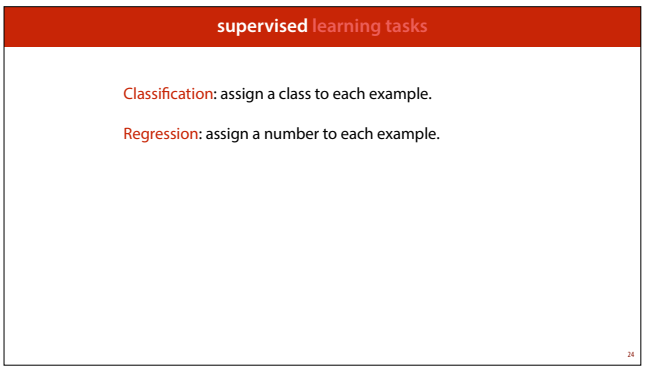
We will discuss Deep Learning, and what makes it so special at various points in the course.

Introduction
Part 2: Classification

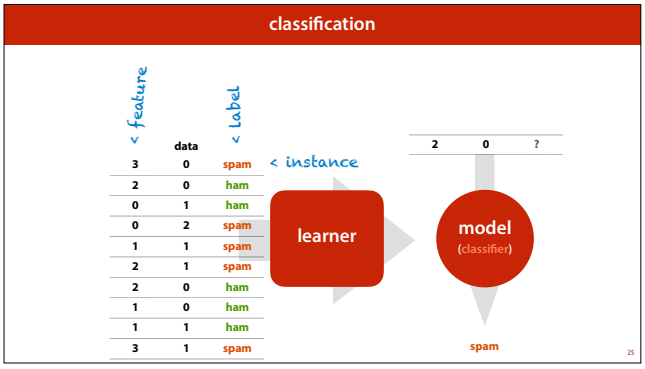
Machine Learning
mlvu.github.io
Vrije Universiteit Amsterdam



In the last video we showed this diagram of how machine learning usually works: we have a problem, we translate a part of that problem to an abstract task, and then we take an existing algorithm for that standard task and implement it.



In this video we will look in detail at the two main abstract tasks for supervised learning: **classification** and **regression**.



This is the basic framework of classification. The data that we provide our system with consists of examples, called **instances**, of the things we are trying to learn something about. In this example, our instances are e-mails.

We must then make a series of measurements about each instance. In the case of e-mails, we may measure how often a specific word occurs. The things we measure are called the **features** of the instance. We can measure numeric features (like age or speed), but they can also be categoric (like gender or color).

Finally we have the **target value**: the thing we are trying to learn. In classification, this is always a

categoric value, or a *class*: one of a handful of possible values. In this case, is the e-mail spam (an unwanted advertising e-mail), or ham (a genuine e-mail).

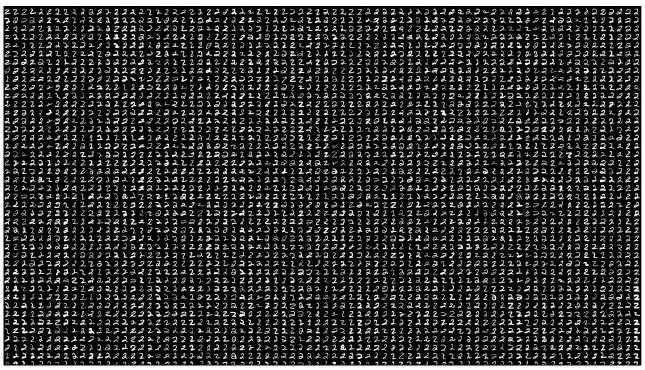
This dataset is then fed to a learning algorithm. This can be anything, but it has to produce a classifier. A classifier is a small "machine" that (attempts) to solve the learning problem. That is, it takes a new instance, one that wasn't in the original dataset, and for which we don't know the target value, and it makes a guess at the target value.

Let's look at some examples of how we can reduce real-world problems to classification. We'll start with **handwriting recognition**. Specifically, reading a ZIP code on an envelope. This involves many difficult problems: aligning the envelope, finding the address, finding the ZIP code within the address, segmenting the address into digits, etc.

Our first step is to reduce the problem to a simple classification problem: we will assume that we are given an image of a *single digit*, and the task is to predict what the digit is. This is a much simpler problem, but still a challenging one. We'll leave all the other problems to other people to solve (either using traditional approaches, or with more machine learning).

The next step is to gather some training data we can learn from.

image source: **https://rafalab.github.io/dsbook/introduction-to-machine-learning.html**

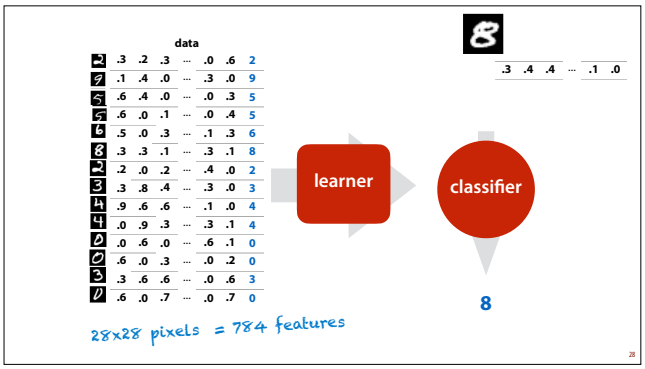


First, we need a lot of examples of digits. This is easy enough with a little clever automation. The second part is more challenging: somebody needs to annotate what digit each picture represents. If we could automate that step, we wouldn't need a classifier, so there's no getting away from the fact that we need to do that by hand.

In the 1980s, researchers at NIST (a US agency) built such a dataset, originally for the purpose of helping the US to evaluate the many digit recognition systems that were become available on the market. This evolved in the the MNIST dataset. It contains 60 000 examples of handwritten digits. This translates very simply to classification: each picture of a digit is an **instance**, and the **target** is one of ten classes: 0, 1, 2, 3, 4, 5, 6, 7, 8 or

9.

The story of MNIST: https://www.youtube.com/watch?v=oKzNUGz21JM

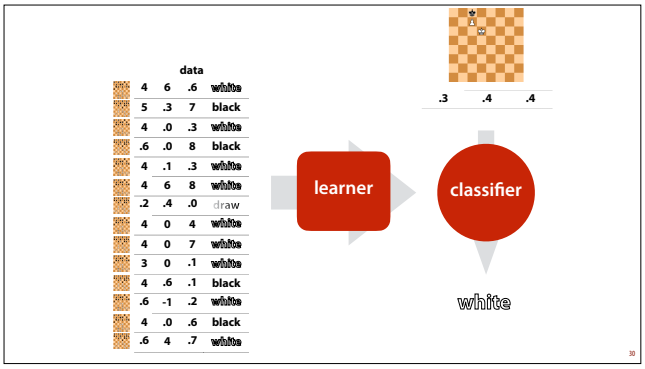


A simple way of attacking this problem is to make each pixel a feature. Here's what that looks like. For each instance, we translate each pixel to a value between 0 (black) and 1 (white). This gives us instance with 784 features each, labeled with a digit from 0 to 9. We feed these to the learning algorithm, which produces a classifier. We then get a new example, and ask the classifier what it thinks. Once we have a classifier that does well, we can use it in a larger system, for recognising digits.

The current best performing classifier for this task has a **probability of 0.21%** of getting an unseen example wrong.

Note that we haven't fully solved the problem of character recognition. We still need to cut a sequence of digits into individual digits, feed those digits to the classifier and process the results. This all the work we have to to to translate our **real problem** to the **abstract problem** of classification. Machine learning has solved part of the problem for us, but there is usually still a lot of engineering left to do.

Let's look a problem that's a bit further removed from classification: game playing. Specifically, playing chess. How do we abstract the problem of playing chess to a classification problem? The trick again is to make things easy for ourselves, by only abstracting *part of the problem*. We won't solve the whole thing with machine learning, but we'll learn a function that'll be useful in a larger chess-playing system.

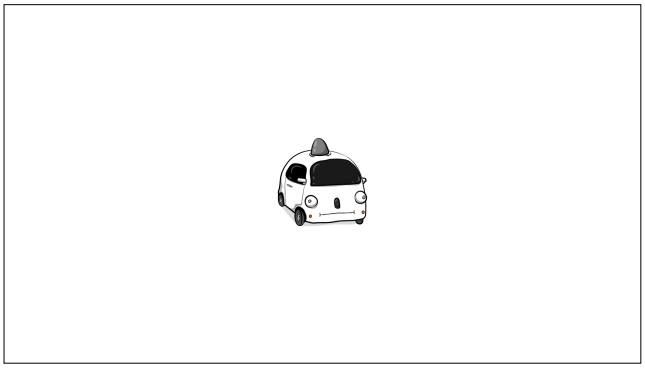


For instance, we could take a database of chess games, and label each position with which player ended up winning the game in the end. The aim is to predict, for given a position, which player is going to win the game.

We could turn such a classifier into a chess player, by searching for positions from which we are likely to win and then playing moves that are likely to lead to those positions. Perhaps you are familiar with the minimax algorithm: you could easily use such a classifier as a *heuristic* in minimax.

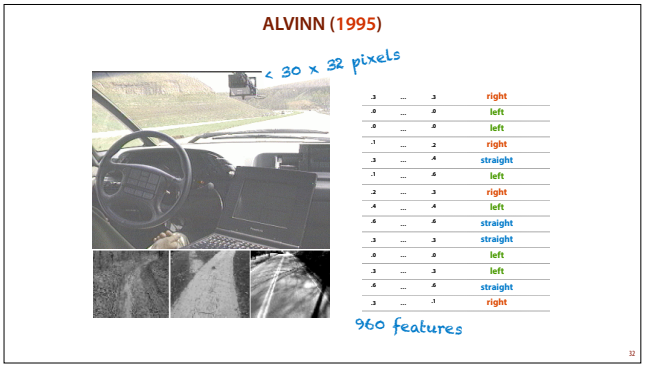
A difficult problem here is *which features to use*. One option is to report how much of each black and white piece is left, which would allow at least some positions to be predicted accurately, if one player has a strong material advantage. For more insightful learning, we need better features. **Domain expertise** can often be translated to good features: are there passed pawns, rooks on an open file, does a player own both bishops, etc.

Again, we haven't solved the whole problem of learning how to play chess, but we've abstracted part of our problem into classification, hopefully making our life a little easier.



Last example, a self driving car. How do we turn the problem of making a self driving car into a classification?

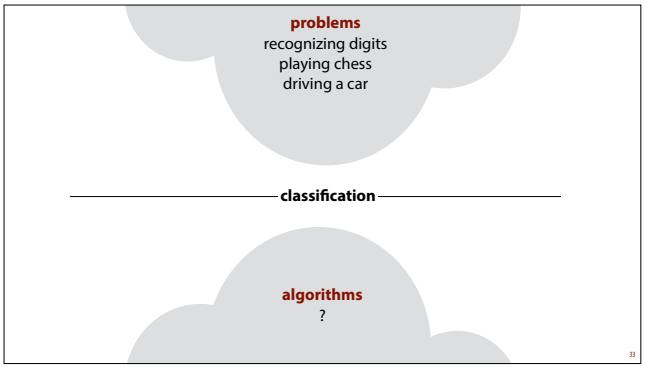
image source: the oatmeal, **http://theoatmeal.com/blog/google_self_driving_car**



Here is an actual self-driving car system from 1995. They used a very low-resolution, black-and-white camera to film the road, and observed a human driver's behavior to label each frame with an action. As with the digit recognition example, we simply make each pixel a feature.

This system actually drove from coast-to-coast autonomously in America (albeit with a human driver executing the system's instructions).

NB: The actual system had more than three actions to allow for more gentle steering.

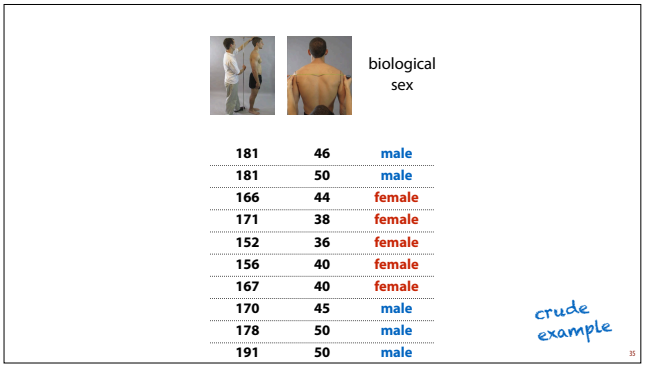


This is what I meant by translating problems into abstract tasks. By translating all these problems into classification problems, we can now apply any **classification algorithm** and see how well it does.

So how do we fill in the other half of this picture? Once we have a classification task, with features selected and a set of good examples, how do we actually produce a classifier?

We'll look at three simple examples: a **Linear classifier,** a **Decision Tree classifier** and a **Nearest Neighbors classifier**. We'll only explain them briefly to give you a sense of how these problems might be solved. Don't worry if you don't totally get it yet. All methods will be discussed in more detail in later lectures.

To explain

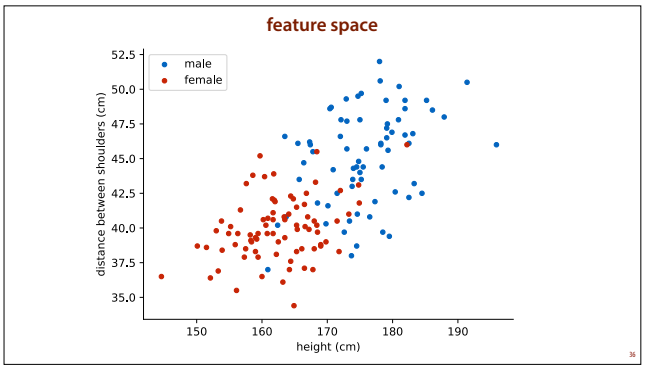


Here's a dataset that we'll use as a running example. Its **instances** are US soldiers, the two **features** are the height (or 'stature') and the distance between the shoulder blades ('interscye'). The class is their biological sex. Are these two values enough to predict whether somebody's sex is **male** or **female**?

This is, of course, an extremely crude example. Biological sex cannot be perfectly predicted from just two measurements, and isn't perfectly captured in these two classes. It's important to understand that most machine learning algorithms are like this: crude approximations. We'll talk more about this problem in the fourth video.

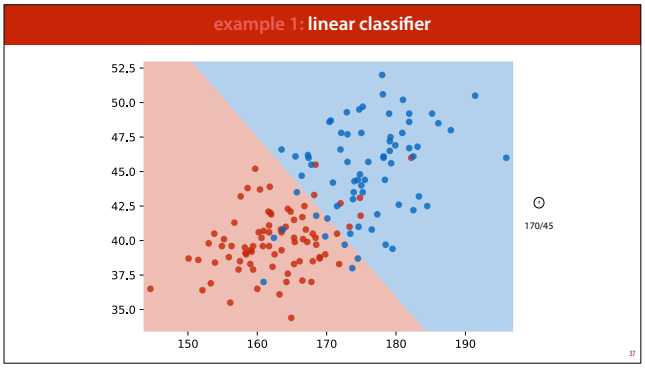
data source: ANSUR II

image source: MEASURER'S HANDBOOK: US ARMY AND MARINE CORPS ANTHROPOMETRIC SURVEYS, 2010-2011



Since we have only two features, we can easily plot our dataset.

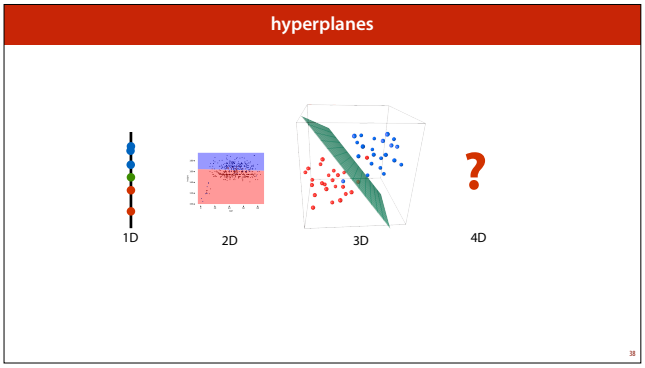
We call this space, where every feature is an axis and every instance is a point, the **feature space**. If we had 3 features, it would be a 3D space. For higher numbers of features, we may have difficulty visualizing the feature space, but that shouldn't stop the classifier: any classification method we come up with should, in principle, work on an arbitrary number of features.



Here is a simple idea for a classifier: *draw a line*. We just draw a line, and call everything above the line **male**, and below it **female**.

As you can see, a few examples end up misclassified, but most of them are on the correct side of the line. Our classifier would already do much better than one that would simply guess at random or call everything **female**.

If we see a new person, all we need to do is measure them, and see whether they end up above or below the line.



An important thing to note is that "drawing a line" is a technique that only works in two dimensions (i.e. if we have two features). Our methods need to work, at least in principle, for whatever number of features we decide to use. The more generic version of the idea to "draw a line" is to cut the feature space in two using a line-like shape.

In 1D, the equivalent structure is a point. Anything above the point is male, anything below it, female.

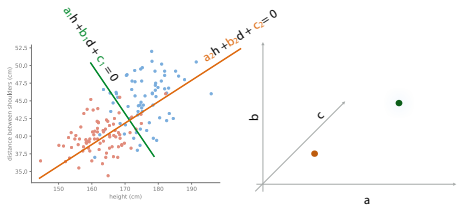
In 3D, we can cut the feature space in two with a plane.

In four or more dimensions, the shape that cuts the space in two is called a **hyperplane**. we can no longer draw it intuitively, but luckily the mathematics are very

simple. We'll see how to define this in the next lecture.

---

the two spaces of machine learning



feature space (2D) model space (3D)

39

Which line should we choose? We can visualise this problem in the feature space. In the feature space (or instance space), each instance is a point, and our current classifier is a line.

Since a line is defined by two parameters (the slope **s** and the intercept **b**), we can visualise the **model space** as a plane. To search the model space, we define a **loss function** which tells us how well a particular model does for our data.

---

loss function

$\text{loss}_{\text{data}}(\text{model})$ = performance of model on the data
(the lower the better)

for classification: e.g. the number of misclassified examples

40

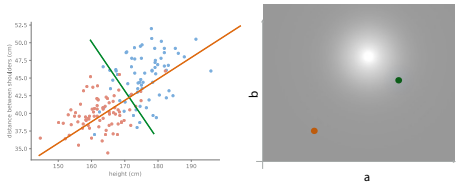
To capture which model we prefer, we define a loss function. The lower the loss, the better the model.

Note that the loss function has the **model** as input and the **data** as a constant (as opposed to the model itself, which is a function of the data).

The best loss function to use for classification is a complex question. We'll come back to that later.

---

the two spaces of machine learning



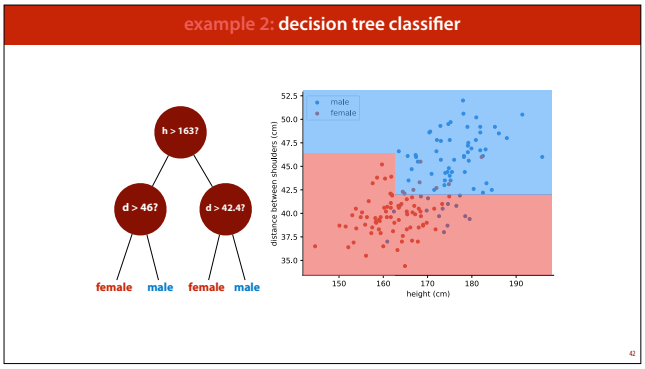
feature space model space

linear classifiers: next lecture

41

Once we have a loss function, we can colour our model space with the *loss of each model* (for our current data). The brighter, the better.

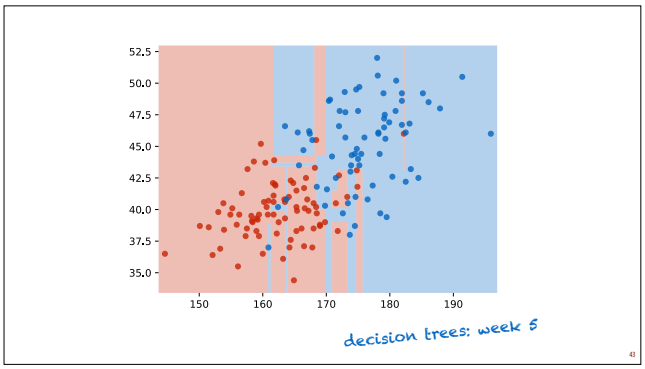
All we need to do now is find the brightest point, which corresponds to the best model. More on that next lecture. In this case we can see which the brightest point is, but remember that the model space is usually high-dimensional.



Here is a radically different approach: **a decision tree**. This classifier consists of a tree, which studies one feature in isolation at every node. In this case, it moves left if the feature is lower than some threshold value, and right if the feature is higher.

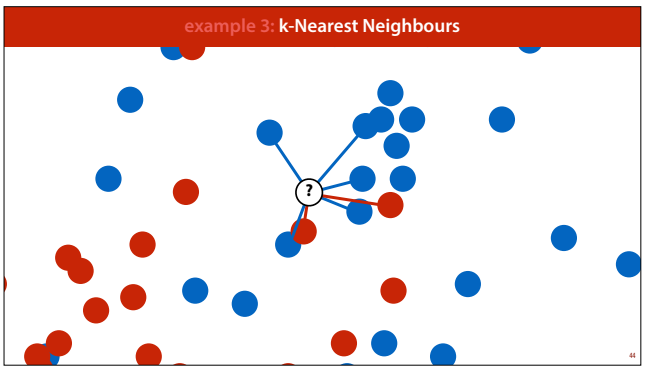
The model space of all possible decision trees is a lot more difficult to visualize. Often, decision trees are "grown" by adding nodes from the root until a particular criterion is reached. We'll discuss how to train decision trees in detail in week 5 (the algorithm is simple, but it requires a little probability theory).

The shape that he classifier draws in feature space to segment the two classes is called the **decision**



Here's what the actual decision tree algorithm from sklearn does (with default settings).

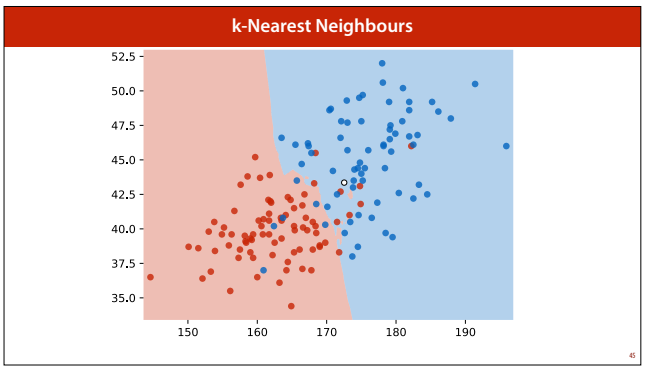
In week 5, we'll see how this learning algorithm actually works.



Finally, here is an example of a *lazy* classifier. k-Nearest neighbours. It doesn't do any learning. It just *remembers* the data.

For a new point (indicated by the question mark), it just looks at the *k* points that are closest (k=7 in this picture), and assigns the class that is most frequent in that set (blue in this case).

k is what we call a **hyperparameter**: you have to choose it yourself before you use the algorithm. We'll discuss how to choose hyperparameters in lecture 4.



Here's what the decision boundary looks like for k=7. The point from the previous slide is indicated in white.

variations

- Features: usually numerical or categorical.
- Binary classification: two classes
- Multiclass classification: more than two classes
- Multi*label* classification: none, some or all classes may be true
- Class probabilities/scores: the classifier reports a probability or score for each class.
  Helpful property for a classifier to have

46

A few variations are possible on this basic scheme. Firstly, the features are usually numerical or categorical. Some models can handle only numerical, in which case, any categorical features have to be translated to numerical ones (we'll see how to do that in lecture 4).

Binary classification (a task with two classes) is probably the simplest and most well-studied task. If you have *more* than two classes, some classifiers (like kNN) can deal with that without a problem. For others (like linear classification), you'll need to find clever way to turn a binary classifier into a multiclass classifier.

Multilabel classification is a much more complex task. We won't go into it in this course, but it's an active subject of research.

Instead of a single verdict, it can often be helpful if a classifier assigns a **score** to each class. If we want a single class, we pick the one with the highest score, but we can also check what the second most likely class is, and how sure the classifier is of its verdict. This is often important if the consequences of of a wrong classification are very serious (i.e. deciding whether to operate on someone, or whether to investigate someone for criminal activity).

---

offline machine learning: the basic recipe

Abstract (part of) your problem to a **standard task.**
Classification, Regression, Clustering, Density estimation, Generative Modeling, Online learning, Reinforcement Learning, Structured Output Learning

Choose your **instances** and their **features**.
For supervised learning, choose a target.

Choose your **model class**.
Linear models, Decision Trees, kNN,

Search for a good model.
Usually, a model comes with its own search method. Sometimes multiple options are available.

47

To summarize: this is the basic recipe for doing machine learning. It doesn't always fit the problem, and we'll look at those cases too. Sometimes your problem doesn't fit very well (reinforcement learning, recommendation), sometime deviating slightly can help performance (sequence learning) and sometimes, taking a radically different approach can turn everything on its head (deep learning).

Nevertheless, if you want to keep things simple, and use existing solutions, **the basic recipe** is a good starting point,

---

in code (cf worksheet 2)

```
# Choose a model: Decision Trees
from sklearn.tree import DecisionTreeClassifier

x_train, x_test = … # matrix of features
y_train, y_test = … # target class labels

tree = DecisionTreeClassifier()
tree.fit(x_train, y_train) # Search for model

# classify new data
y_predicted = tree.predict(x_test)
# see how well we do
print accuracy_score(y_predicted, y_test)
```

48

Here's what the basic recipe looks like in code (using the `sklearn` library). Note that the actual machine learning happens in just two lines of code.

All you need to do is decide your features, and decide your target values (classes in this case). Once you've done that, you're doing machine learning in two lines of code. You can then test how well your model does (more about that in week two) and keep trying different models until you get the performance you're happy with.

That's all we'll say about classification in this lecture. In the next video, we look at regression, and some other abstract tasks.

image source: **https://twitter.com/archillinks/status/1022889384494940160**

Introduction
Part 3: Other abstract tasks

Machine Learning
mlvu.github.io
Vrije Universiteit Amsterdam

---

abstract tasks

supervised

- classification
- regression

unsupervised

- clustering
- density estimation
- generative modeling

51

We've looked at classification, as our first example of an abstract task. In this video. We'll see some others. First up: regression.

---

regression

data		
3	0	1.1
2	0	4.5
0	1	12.3
0	2	5.1
1	1	9.1
2	1	1.2
2	0	5.2
1	0	6.1
1	1	1.9
3	1	1.8
6	0	3.2
0	1	0.1
0	0	0.4

learner

$\mathbf{x}_i$ features of instance i: 2 0 ?

model

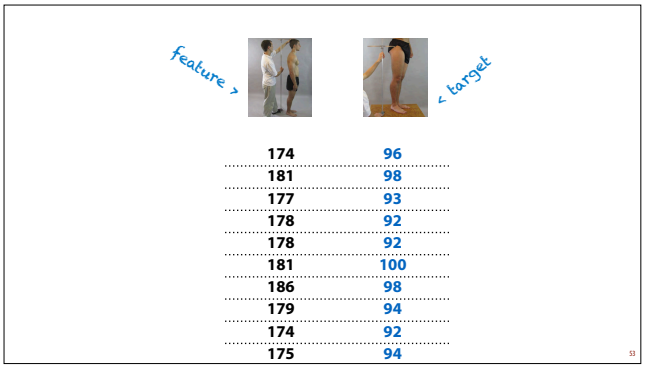
$f(\mathbf{x}_i)$ model

1.8

y_i true label for x_i

52

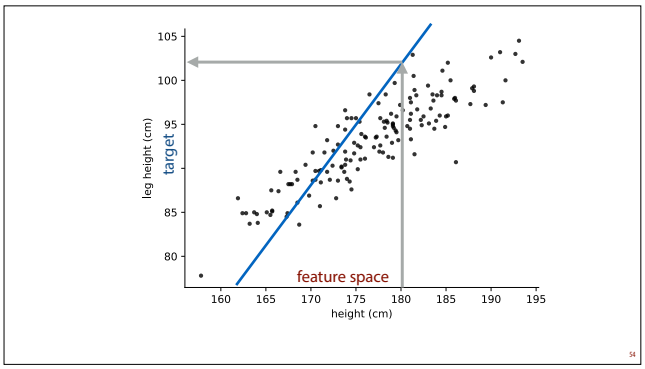
Regression works exactly the same as classification, except we're predicting a number instead of a class. That is, the model we're trying to learn is a function from the feature space to $\mathbb{R}$.



174	96
181	98
177	93
178	92
178	92
181	100
186	98
179	94
174	92
175	94

Staying in the same domain, here is another dataset. Note that we have **only one feature** this time. The other numerical column is the **target** data. For general regression we should be able to handle datasets with arbitrary numbers of features.

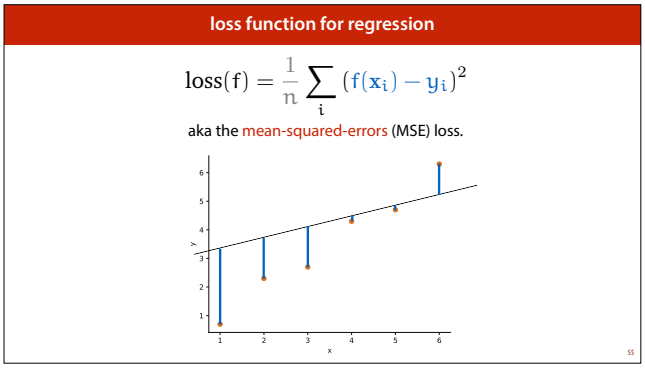
data source: ANSUR II
image source: MEASURER'S HANDBOOK: US ARMY AND MARINE CORPS ANTHROPOMETRIC SURVEYS, 2010-2011



Here's what our data looks like. Note that though it looks the same as in the classification example, this time we're plotting both the targets and the feature space in the same figure.

We can use a **linear model** again. But note how differently we're using the model. Previously, we wanted to segment the feature space into two classes. Now we're trying to model the relation between the feature(s) and the target. The model has the same shape, but we're using it very differently.

The line I've drawn here isn't very good. It predicts much too high a value. To determine how good a model is, we must again choose a **loss function**.

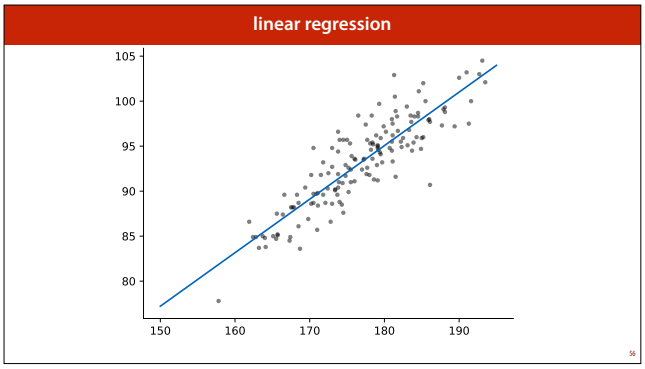


The **loss function** maps a model to a number that expresses how well it fits the data (the smaller the loss, the better). Offline machine learning boils down to finding a model with a low loss for your data.

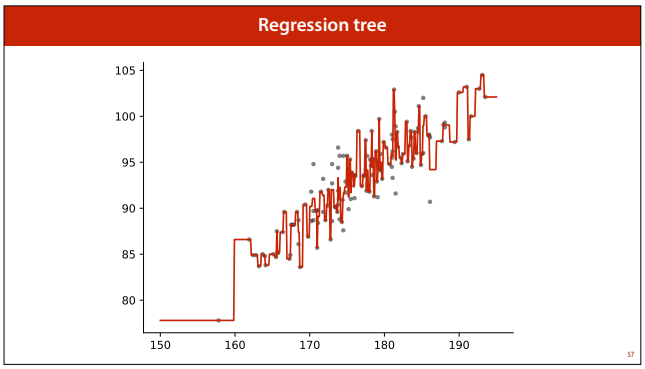
Here is common loss function for regression. **p** stands for the parameters that define the line. We simply take the difference between the model prediction and the target value from the data. This is called a **residual**. We square, and then sum all residuals.

Sometimes we also divide by the size of the data (n). Occasionally, we multiply by 1/2 for technical reasons. This changes the actual value, but not *which* model produces the lowest loss, which is the question we want to answer.

image source: **http://cs.wellesley.edu/~cs199/lectures/35-correlation-regression.html**

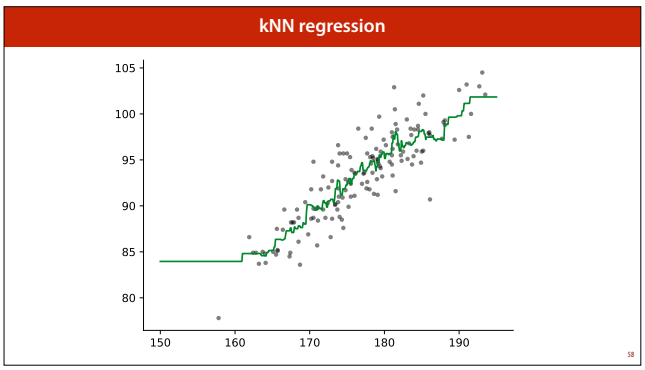


This is the line with the lowest MSE loss for this data.

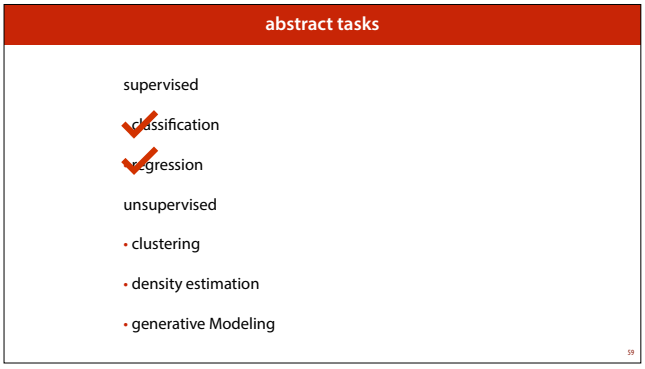


We can also use the **decision tree** principle to perform regression, giving us a **regression tree**.

We simply segment the feature space into blocks, using a tree as before, and instead of assigning each a *class*, we assign each a *number*. This model covers the data very well, for most points in our data it predicts exactly the right value. Does this make it a rally good model?

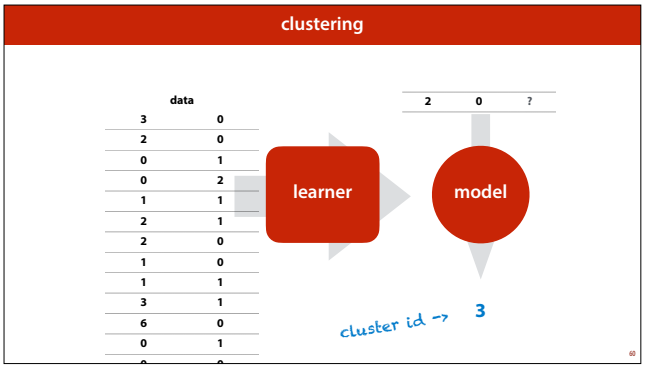


For the sake of completeness, here is what kNN *regression* looks like. Its prediction is the average of the k nearest points in the data (k=7 here).



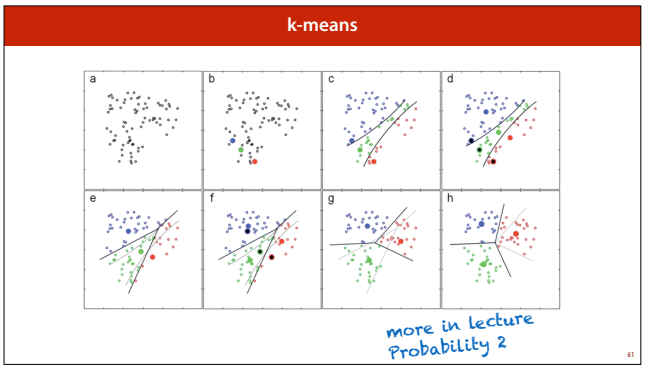
Next up are the unsupervised tasks. In classification and regression each instance came with a **label**: an example of the sort of output we wanted our model to predict for each input.

In unsupervised tasks, we have only the inputs. The task for the model is just to find any useful structure in the data.



In the case of clustering, we ask the learner to split the instances into a number of clusters. The number of clusters is usually given beforehand by the user.

This looks a lot like classification, but note that there are no example classes provided by the data.

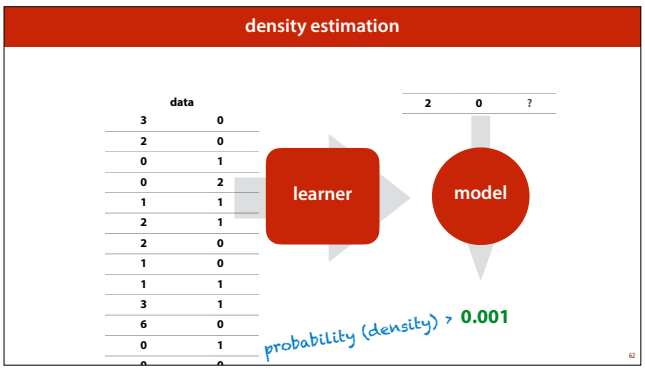


Here is one example of a simple clustering algorithm.

This algorithm is called **k-means** (not to be confused with kNN). In the example we will separate the dataset shown in (a) into three clusters.

We start (b), by choosing three random points in the feature space (the red, green and blue points), called the "means". We then colour every point in the colour of the nearest mean.

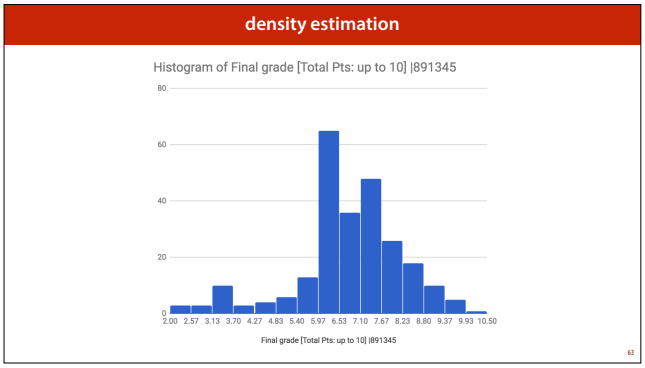
The second step is to remove the original means, and recompute them as the actual means of the sets of red, green and blue points. We then iterate these two steps, alternately recomputing the means and recovering the points, until the algorithm converges to a stable state.



In density estimation, we want to learn how *likely* new data is. Is a 2 m tall 16 year old more or less likely than a 1.5 m tall 80 year old? We predict a number for each instance, and that number expresses how likely the model thinks the given instance is.

The number that the model produces should be a *probability* if the feature spaces is discrete. That means that the sum of all answers over the whole feature space should be 1.

If the feature space is continuous (numeric features), the answer should be a probability *density* (and all answers should *integrate* to 1).



That may sound abstract, but density estimation is probably the machine learning task that most people have already done before.

Density estimation is the task of modelling the *probability distribution* behind your data. Most of you will have fit a distribution to a dataset at some point.

Here is an example: the final grades from 2017. If you know a bit of statistics, you can probably see sort of a normal distribution in this. Once you've fitted a normal distribution, you can give a density estimate for any grade.

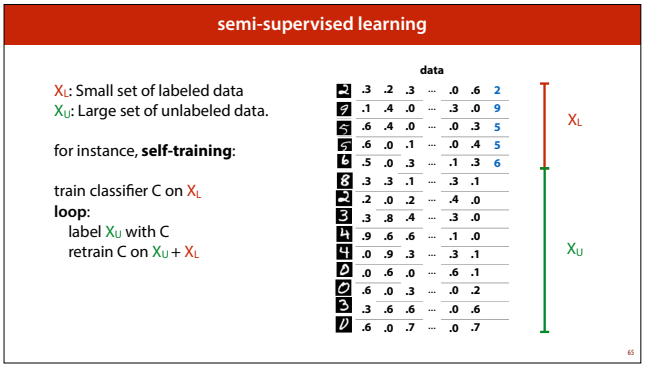
Except that in this case, there are three peaks. these could be explained by noise, but we could also fit a

mixture of three normal distributions to this data, to explain the peaks. This is a much more difficult task, to which we will return in a few weeks. For now, the lesson is that for simple models like a normal distribution density estimation is so easy it's not usually seen as machine learning, but as the models get more complex, the task gets more complex also.

With complex data, it's often easier to *sample* from a probability distribution that it is to get a probability (density) estimate. Building a model from which you can sample new examples is called **generative modelling**.

These people *don't exist*. These pictures were *sampled* from a model trained on a large dataset of images of faces. Note that this is not a 3d model, or a generator that started with a basic face and filled in the details: all the model saw was a large collection of pictures. This is a typical example of the power of **deep learning**, which we will discuss in the third week.

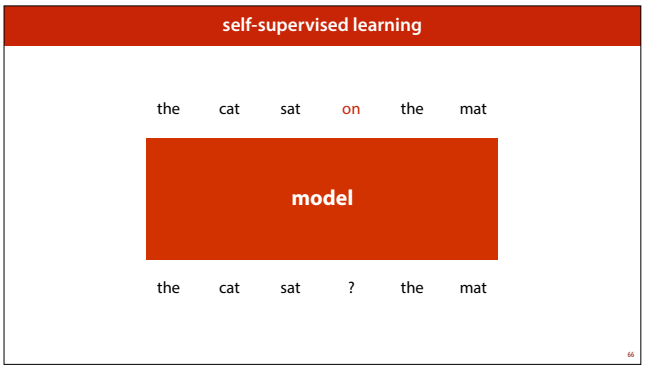
image source: https://arxiv.org/abs/1812.04948



In many cases, unlabeled data is very cheaply available, while labeled data is expensive to acquire. In such cases, semi-supervised learning can be useful: this involves learning from a small labeled set and a large amount of unlabeled data.

A simple example is self-training: we train a classifier on the unlabeled data and use it to complete the dataset. Then we train on the full data and repeat the process. From this example, it's slightly mysterious why this should help. For now, we'll just say that the classifier trained on the whole data can better understand the basic structure of the instances, and then attach the label based on that deeper understanding of the structure. If that doesn't feel like

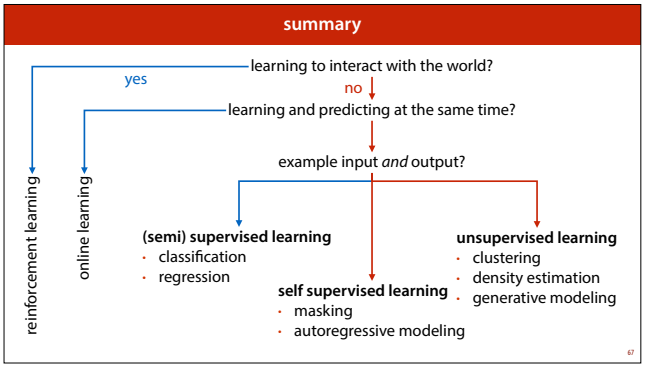
a satisfying explanation, you'll have to wait until later in the course, when we can talk about this in greater depth.

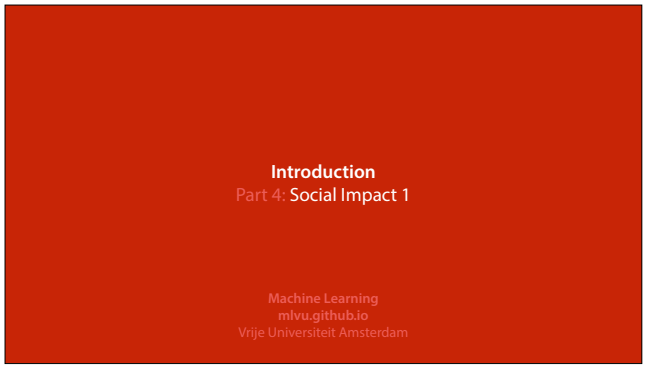


Recently people have been referring to a family of methods as **self-supervised learning**. It refers, generally to ways in which a large dataset can be used to train a model without requiring annotation, in such a way that no or little annotation is required.

One example is in the domain of natural language, to feed a model that can read and produce sequences, a sentence with on or more words masked out. This is not in itself a useful task, but it is a task that can be performed on unlabeled data. And, a model that can learn to do this well, has likely learned a lot about the structure of sentences, which means it can then be used to build on for other, more useful tasks (possibly with a small amount of labeled data thrown in).

Semi supervised learning and self-supervised learning have a lot in common, and it's not quite clear where one begins and the other ends. In general, self-supervised learning refers to deep learning models, and to clever training schemes using unlabeled data. We'll see some more examples when we start talking about deep learning.





Throughout the course, we'll occasionally stop and look at the impact that this kind of technology has on society. This is rapidly becoming more important as machine learning is being rolled out at national and international scales.

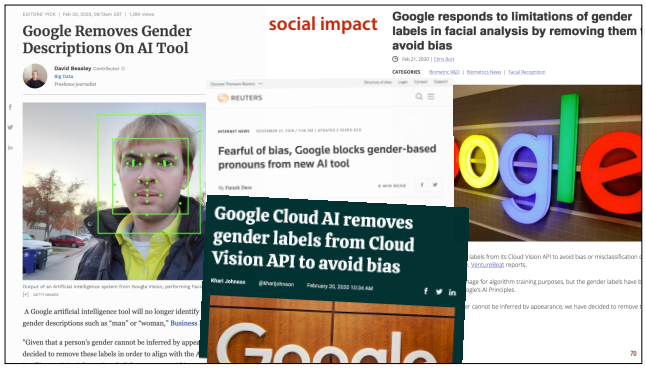
Sometimes we'll do this as part of the regular lectures, and sometimes, we'll create a separate video to focus on some important aspects. In this video we'll look at some of the questions it's important to ask of machine learning systems and machine learning research.

A quick disclaimer: the question of social impact is difficult to divorce entirely from one's personal values. I am a secular, left-wing progressive when it comes to most issues, which will no doubt come through in these videos. I have done my best to present the discussion rather than the conclusions, and to present mostly facts that are difficult dismiss if you believe in basic academic investigation. You are entirely free to form your own opinion, of course, and the exam will only test your knowledge of the material presented here.

In the first video, we saw one of the earliest examples of machine learning: a perceptron being used to classify the sex or gender of a person by their photograph. We've used this example to illustrate how classifiers work, and we may return to it occasionally.

This may seem like a harmless example, and indeed for a long time the exercise was pretty academic. Machine learning simply didn't work very well yet, and this problem gave us a nice balanced set of classes, and a difficult computer vision task that we knew could in principle be solved with good accuracy (because we can do it).

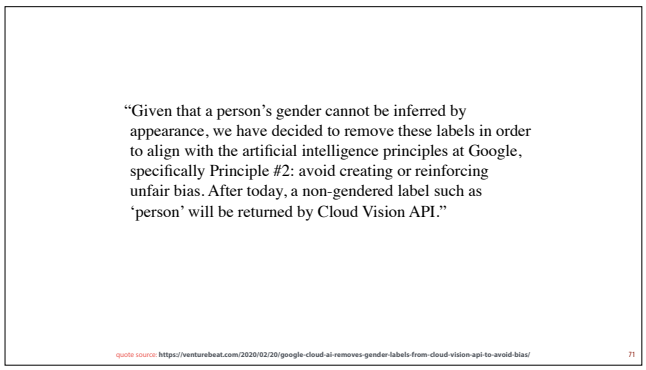
In short, it was a good benchmark with which to study our models. Now that we have effective models, we need to consider what it means to actually use those models.



However, now that we have actually "solved" the task (that is, we have machine learning systems can do it as well as humans can), we need to look at what the impact is when we actually start using such systems in the real world. Sex or gender detection may seem harmless at first sight, after all, it's something we all do subconsciously hundreds of times every day. But actually, building automated systems for it is highly controversial. So controversial in fact, that Google has disabled the option in its Cloud vision API.

This is a lucrative product for Google, and a reasonable guess of a subject's gender is likely to be useful and something that occurs frequently. If google removes such a feature from their product, they must have a compelling reason.

In this video, we will look at this example in detail, and consider the different reasons that people offer why we shouldn't build such systems, even though we can.



First, let's see what google offered as an explanation for removing the feature. Their argument centers on the impossibility of inferred gender from physical attributes.

But is this the whole story? We cannot perfectly infer a traffic sign or a digit from an image, but but we can make a pretty good guess. In fact these days, guessing a person's sex or gender or sex can be done with pretty high accuracy compared to most machine learning tasks.

So the fact that it can't be done perfectly surely can't be the whole story: that is true for almost all machine learning applications, and any label returned by the cloud vision API. What makes gender special? Why should gender only be used if it can be perfectly inferred.

To get to the real reason that gender classification is controversial, we need to look more carefully at the problem.

sensitive attributes

As features or targets. Examples:

- Sexual orientation
- Race, ethnic identity, cultural identity
- Gender and/or sex

72

The first part of the problem is that gender and sex are examples of what we'll call **sensitive attributes**: features or targets associated with instances in the data, that require careful consideration. These are some examples of sensitive attributes, but many more exist.

What makes an attribute sensitive?

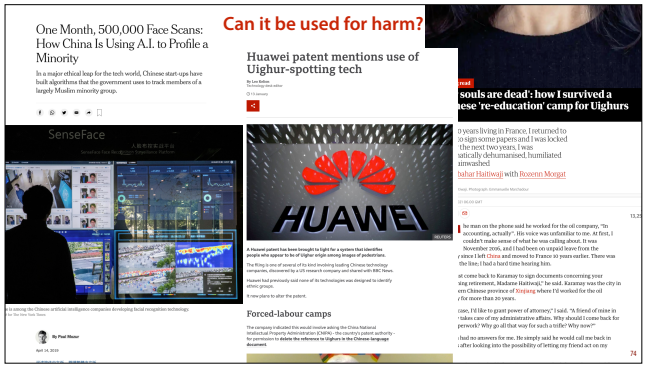
Can it be used for harm?

Can mischaracterizing relations become offensive?

Is it commonly used to *discriminate*?
Explicitly, as in apartheid regimes, or implicitly, through structural inequality.

73

To decide whether or not an attribute is sensitive, and if so, how it should be treated, we can ask ourselves several questions. In this video, we'll focus on the following three.



The first, and perhaps most obvious question to ask yourself, is can my model be used by people who are explicitly seeking to do harm. For instance, the Chinese government is on a large scale, subjecting people of Uighur ethnicity to heightened surveillance, and incarceration under conditions that violate human rights.

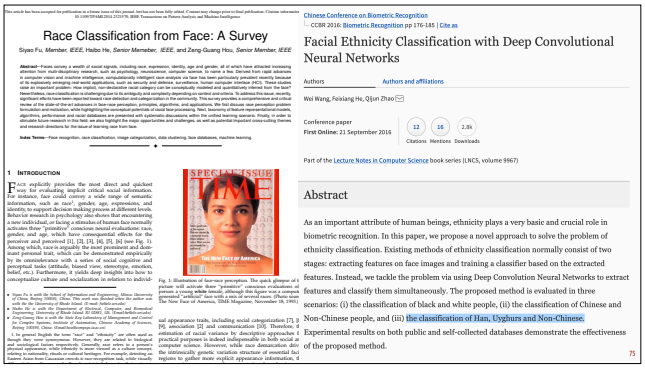
The Chinese government, of course, doesn't characterize this as causing harm, making their case on arguments of national security.

sources:

https://www.nytimes.com/2019/04/14/technology/china-surveillance-artificial-intelligence-racial-profiling.html

https://www.theguardian.com/world/2021/jan/12/uighur-xinjiang-re-education-camp-china-gulbahar-haitiwaji

Fu, Siyao, Haibo He, and Zeng-Guang Hou. "Learning race from face: A survey." IEEE transactions on pattern analysis and machine intelligence 36.12 (2014): 2483-2509.



One part of this system is classification of ethnicity, based on images of people's faces (sometimes explicitly images captured by face recognition cameras). It's quite common to see research emerging from Chinese institutions focusing on the problem of ethnicity classification, with Uighur ethnicity explicitly used as a class.

Even if we want to leave the question of what constitutes a harmful effect aside, we can say that the effect of this technology, which most people and organizations in the west consider harmful, is intended and explicit.

That is, everybody can agree that this technology will make it easier to track people of a given ethnicity. The discussion is over whether that should be done.

In other cases, such effects are not explicitly intended by the makers of the technology. In this article, the organization ProPublica broke the news that a system used nation-wide in America to aid parole decisions was considerably more likely to deny black people parole than white people, even when all other factors were accounted for.

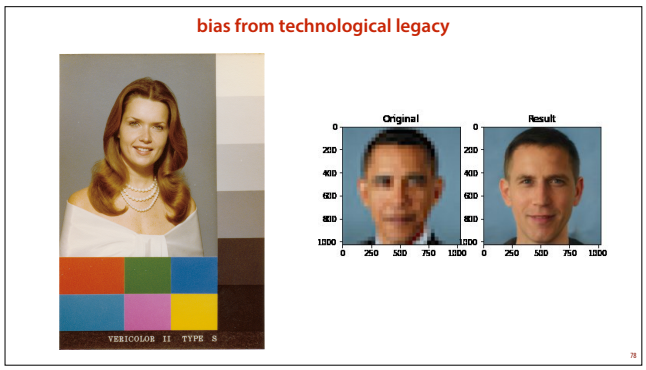
This was not an explicit design choice of the makers of the system (a company called NorthPointe). In fact, they explicitly excluded race as a feature. However, even if we exclude sensitive attributes as features, we can still *infer* race from other features. This means that if we include a few features like postcodes, the system can still make the classification it would have made if race had been available.

The question of where this disparity comes from is an important one. We'll look at this in more detail in the third social impact video.

An important source of bias is the distribution of the training data. Where we get our data has a tremendous impact on what the model learns. Since machine learning often requires large amounts of data, we usually can't afford to control the gathering of data very carefully: unlike studies in life sciences, medicine and so on, we rarely make sure that all variables are carefully controlled.

The result are systems that have unexpected biases. This is a picture of Joy Buolamwini. As a PhD student she worked on existing face recognition systems. She found that if she tested them on her own face, they would not recognize her, and she needed to wear a light-colored mask to be recognized at all.

One aspect of this problem is training data bias. If we gather data carelessly from the internet, we end up inheriting whatever biases our source has. If white people are disproportionally represented, then we end up training a system that works less well on non-white people.



The problem is compounded by the way technologies build on one another. On the left is one example: the Shirley card. Such images (named after on of the models on the first one) were used by lab technicians to develop color photographs. This image was taken as a reference calibrate photo printers in small labs. This shows that white skin was, for a long time the main target in developing photographic technology.

Eventually, other test cards were developed, but not before color film and development had ben focused on white skin for decades. Since digital photography was largely developed to mimic film, it's quite possible that some of these biases are still present in modern day technology. Certainly most photographers will tell you that capturing black skin well is a skill in itself [1].

A more recent example is the PULSE system [2]. This is a rather ingenious method for generating reasonable high-resolution versions of low resolution photographs. Interestingly, the method doesn't require any training data of its own: it relies on an existing generator network (which we'll explain in lecture 9). Unfortunately, the generator network they used (known as StyleGAN) turned out to be biased (most likely a result of the data it used). The result was that people found that images of non-white people were upsampled to white people.

[1] **https://www.anothermag.com/art-photography/12799/antwaun-sargent-joshua-**

kissi-in-conversation-just-pictures-exhibition-st-louis

[2] **https://github.com/adamian98/pulse**

Finally, how you choose to **use the predictions** of your model can amplify bias, even if the predictions themselves are in some sense correct.

Shown here is google's machine translation system. A sentence which is gender-neutral in English, like "My friend is a doctor" cannot be translated in a gender-neutral way into Spanish. In the earlier versions of Google Translate, a gender was chosen (implicitly), mostly dictated by the statistics of the dataset. You may argue that these statistics are in a sense reflective of biases that exist in society, so that it is indeed more likely that this sentence should be translated for a male.

However, that doesn't mean that we're *certain* that the user wants the sentence translated in this way. We might build a model that predicts that this sentence should be translated with a male gender with 70% probability. Let's assume for the sake of argument that that probability is entirely correct.

That *prediction* may be entirely correct, but that doesn't tell us anything about what the correct *action* is. If we simply pick the gender with the highest probability, we're actually *amplifying* the bias in the data.

The solution (in this case) was not to reduce the uncertainy by guessing more accurately, but to detect it, *and communicate it to the user*. In this case, by showing the two possible translations to the user. The lesson here is that even if your predictions are sound, designing the correct **action** is still a difficult challenge.

Are you predicting what you think you're predicting?

MISCELLANY

The questions "Have you ever used Derbisol?" and "How often?" sometimes appear along with questions about alcohol, cocaine, and marijuana use on youth-risk surveys for students. Derbisol is a fictitious drug devised to test the reliability of the responder. In one survey, 163 of 894 students said that they had tried Derbisol—or 18.2 percent.

source: https://www.laphamsquarterly.org/intoxication/miscellany/have-you-ever-used-derbisol

80

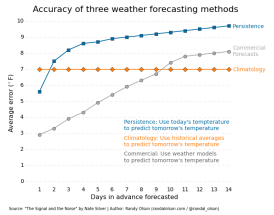
It's also important to note that the target variable may not always be saying what you think it's saying.

This is a common problem with **self-reporting**. If you ask respondents for some value you're interested in, rather than testing it directly, you often end up with inaccurate results. Either because people are lying to you, or because they simply don't have an accurate idea of what you're interested in.

Imagine a survey where measures like this aren't taken, and we trust the students at face value when we ask about their drug use. If we then train a classifier to predict drug use from a set of features like extroversion, social background, or education level, we may think we've found a link between drug use and these features, when actually what we've found is a predictor for how willing people are to lie on a questionnaire.

It matters what you're predicting *from*.

Persistence: the weather forecasting tactic of predicting today's weather as tomorrow's.

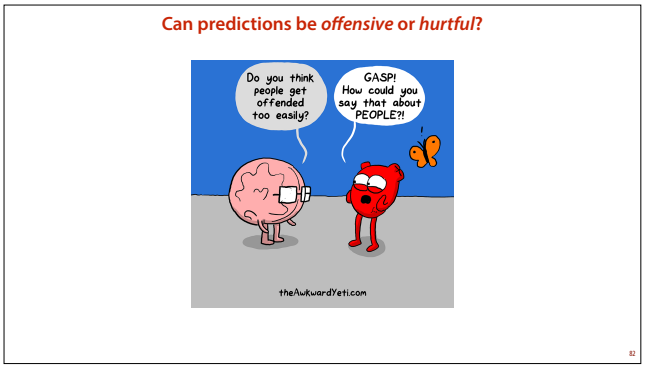


source: **http://www.randalolson.com/2014/06/21/we-can-only-forecast-the-weather-a-few-days-into-the-future/**

81

Another question is what features are you looking at for your predictions. One very effective for predicting the weather is simply to look at the weather today and to make that the prediction for tomorrow. This is surprisingly effective, and you need a very sophisticated model to do any better.

Nevertheless, for many use cases, this is an entirely unsuitable prediction. In short **accuracy isn't all that matters**. A ship's captain, for instance, will be interested in the probability of a storm. For this particular, rare event the persistence model works terribly even if it works well on average. The captain will be much more interested in a model that looks at relevant features to warn them of even a small probability of a storm than a model that simply predicts sunshine if today was sunny. Part of the issue if that the captain is looking for predictions from informative features like mounting or falling air pressure.



Finally, it's important to ask whether predicting a particular attribute can be offensive or hurtful.

This is a more nebulous question than the others. Whether or not something causes offense is highly subjective. Causing offense is not illegal, and indeed it has often been instrumental in improving society. Many people feel that being offended is too often used as a shorthand to shut down meaningful conversation. Perhaps it's better to consider the questions of whether your predictions are hurtful.

Either way, when you build a product that behaves intelligently, it's worth thinking about whether you want its automated behavior to be offensive. Especially if you're rolling that behavior out to hundreds of thousands of users.

As a simple example, imagine if we spoke at a party and I guessed your sexuality, and told you I was doing so based on the shape of your nose. You might be offended (regardless of whether my guess was right or wrong). Even if there is a broad correlation between nose shape and sexuality that I could use in my defense, it would probably still feel to you like I was taking a deep and complex aspect of your identity, and reducing it to a simple thing, to be guessed at.

It's not easy to pin down quite exactly where the offense comes from. My best guess is that it's not so much the method of guessing that I chose to employ, but *the fact that I felt it necessary to do so at all*. I could have asked and been certain, or I could simply have left it. Since the attribute is a sensitive one, it deserves a sensitive course of action. In short there's a difference between being able to make a crude guess, and choosing to do so.

In designing computer systems, a good rule of thumb is that if a behavior is not acceptable in a social context for people, then users will get upset if a computer system does it. Consider a website that asks you to give your email before is shows you its homepage. That's like a person who asks you intimate questions before even introducing themselves. Computers should follow social norms, which includes not guessing at sensitive attributes.

Again, everybody is free to behave offensively, but the consequences are theirs. If you don't treat people's sensitive attributes with respect, don't be surprised if they get upset.

Should we include sensitive attributes in data at all?

To study bias, we need these attributes to be annotated.

If we remove them they may be inferred from other features.
Postcode, shopping habits, profile picture.

Directly using a SA may be preferable to indirectly doing so

There are valid use cases
Race and sex affect medicine. Often requires a *causal* link.

83

If you agree that these attributes are sensitive, you may ask whether we should include them in our data at all?

This all depends on context. In some cases, we may want to study whether racial or gender bias exists. In such cases, we need the data annotated.

Removing sensitive attributes doesn't mean that we cannot discriminate on them. Many features are correlated with the sensitive attributes, so that algorithms can still infer the sensitive attribute, and then produce a biased result based on that. In such cases, it may be preferable to include the sensitive feature explicitly and with the user's consent so that we have more control over how it is used, or we can at least explain it better.

Finally, there are often valid use cases where we *can* use sensitive attributes in a responsible way. For instance, medical results are highly dependent on sex and race (and sometimes even sexuality). In conditions that are difficult to diagnose, like Parkinson's these may be crucial factors to consider.

Should we stop using them as targets?

What is input and what is target is not always clearly separated.
Embeddings, clustering, semi-supervised learning, link prediction

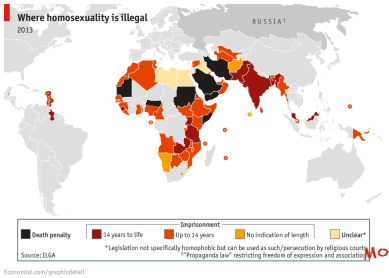
Showing that sensitive attributes *can* be inferred, may serve as a warning to those who are vulnerable.
Building a proof-of-concept in a controlled setting is sometimes the best way to warn the world that something can be built.

84

So, if we cannot just decree that they should never be used as features, can we perhaps agree that they should never be used as targets?

Unfortunately, this is also not an easy call. Many algorithms work without explicit targets. Often, these learn representations that can be used to predict *all* information in the data including the sensitive attributes.

Also, building a system that explicitly does what we consider harmful, may be an affective way to warn people of which dangers exist.



More in social impact 2

85

In the next video we'll look at a classifier that was built to predict a person's sexuality from profile pictures on a dating site. The authors' stated intent was exactly this: to warn people that they may be exposing sensitive information purely by putting their image online.

In eight countries in the world, homosexual acts carry the death penalty. If facial features *are* correlated to some extent with sexual orientation, and law enforcement in these countries is incentivised to find as many non-heterosexual people as possible, then it's important for gay people people to know that just by putting a photograph online, they may be exposing themselves to a higher degree of scrutiny.

sensitive attributes

use with extreme care

- consider user communication over prediction

check the distribution

do not:

- imply causality
- overrepresent what your predictions mean

86

social impact

So, let's return to our gender classifier, and ask some of these questions. Is sex or gender a sensitive attribute and if so, what should we do about gender classification?

Can mischaracterization be offensive?

Sex

- Physical characteristics at birth
- About 0.02% of people outside the male/female classification (intersex)

Gender

- Pyschological identity
- About 0.4% percent outside the male/female classification (e.g. non-binary)

not a perfect distinction

Transgender

- Gender dysphoria: psychological distress due to difference between sex assigned at birth and gender.
- About 0.6% *openly* transgender people in US.
- Cis-gender: matching sex and gender

88

In discussing these matters, it is helpful to make a distinction between the type of body a person is born with, and their psychological gender identity. The first is usually referred to as sex, and the second as gender. These are not perfect distinctions, and it's often not clear whether we are talking about sex differences or gender differences. But they serve to illustrate the basic problem.

People whose sex and gender do not match can suffer considerable mental health problems when living according to their original sex rather than their gender, leading to extremely high rates of attempted suicide (40% vs a 4.6% national average)[1]. It is well-accepted in the psychological community that living with gender dysphoria is extremely distressing and that in adults, the best course of action is to conform to the gender rather than the sex.

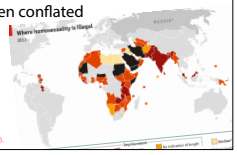
NB: Sometimes the phrase transgender is used to cover both people whose sex and gender differ and people who do not identify as either male or female. For our current purposes, it is helpful to keep these categories distinct.

[1] The report of the 2015 U.S. Transgender survey, NCTE 2015, **https://transequality.org/sites/default/files/docs/usts/USTS-Full-Report-Dec17.pdf**

Can it be used for harm?

In only a handful of countries are trans persons explicitly criminalised, either through a piece of legislation or religious law or edict (which often have the force of law) and are easily classified as so-called "cross-dressing" laws. This is the case in the following thirteen countries: **Brunei, the Gambia, Indonesia, Jordan, Kuwait, Lebanon, Malawi, Malaysia, Nigeria, Oman, South Sudan, Tonga,** and **the United Arab Emirates**. Meanwhile, although **Iran**'s Islamic Penal Code is slightly more vaguely worded in this respect, its impact is no less severe on people who transgress gender norms in their gender expressions.

Mostly considered minor offences, but often conflated with homosexuality.



source: ILGA World: Zhan Chiamet al. *Trans Legal Mapping Report 2019: Recognition before the law* (Geneva: ILGA World, 2020).

In thirteen countries

Can it be used to counter harm?

Inferring sex for medical applications

Countering bias, to increase representation

Studying bias in historical data

Studying the correlations that other ML algorithms make implicitly

90

Can mischaracterization be offensive?

Difficult to understand with no experience of gender dysphoria

Pay attention to the features:

- Physical features predict biological sex.
  They are highly *correlated* with gender, due to a high majority of cisgendered people.
- Other features (dress, grooming) *may* predict gender over sex.

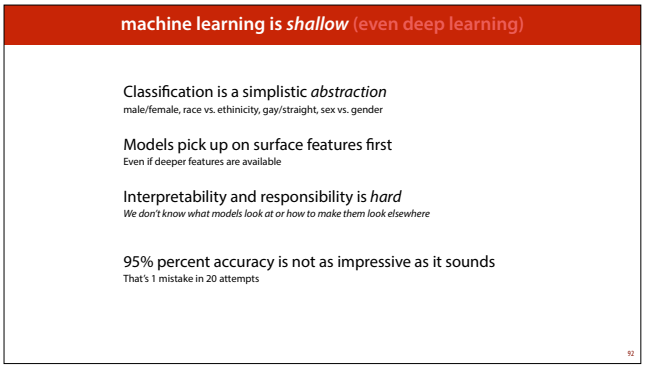
How are your predictions used?

- Targeting ads, Policing
- What are you *implying* about causality and accuracy?

91

If you've never had any experience of gender dysphoria, it can be difficult to understand how intense the experioence can be and how much it can hurt if somebody treats your gender lightly. The best we can do as system builders is to consider the evidence (remember the attempted suicide rates) and listen to our users.

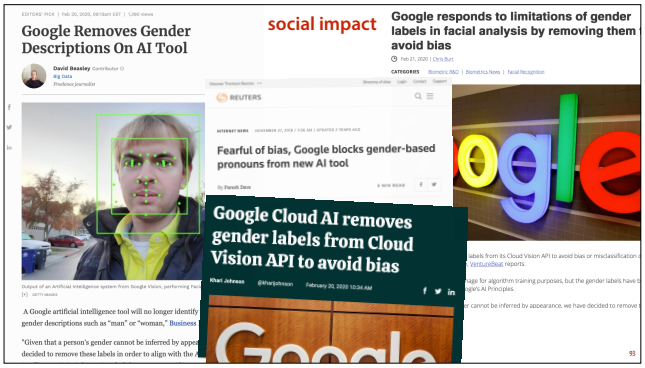
Where we cannot escape simply asking users for their gender when that is apt, or doing without, and we have a good reason to predict it, we should consider carefully which features we use: do they predict gender or sex. One may be correlated with the other (because a large majority of people are cis-gendered), but carelessly using your predictions may be seen as implying causal links that aren't there.



The problem is not solved, and it may never be. It's a social problem more than a computer science one. The important thing to remember, is that we as builders of systems that are deployed at scale, have a responsibility to consider the impact of the decisions we make. In particular we should always keep in mind how shallow machine learning is in its thinking (even if we use *deep* learning).

Classification is a particularly strong example. We like classification as an abstract task, because it's a setting in which our models are easy to evaluate and to train. But that should not blind us to the fact that usually contraining a phenomenon like gender or sexuality to a small set of categories blinds us to the complexities of the thing we are actually trying to predict.

Imagine a classifier that predicts the genre of a movie as romance, action or comedy. In a research setting, this is a very nice, simle way to test our models, and see what they can do. But if we move to a domain where we are actually interested in saying something about movies, this categorization is woefully inadequate. It's when we're investigating our models, but it's terrible to actually use in production settings (i.e. when we want to take actions rather than make predictions).

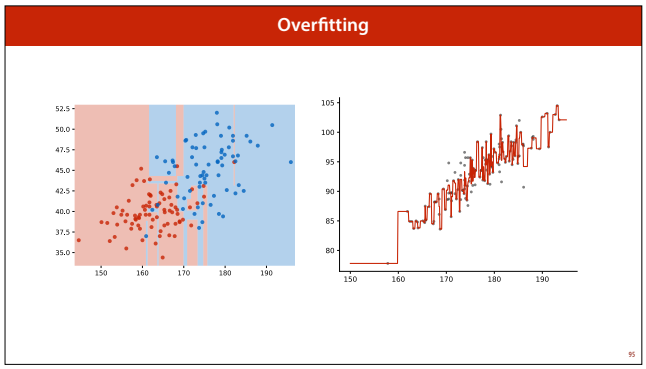


So, this hopefully explains to some extent why GGoogle made the choice that it did.

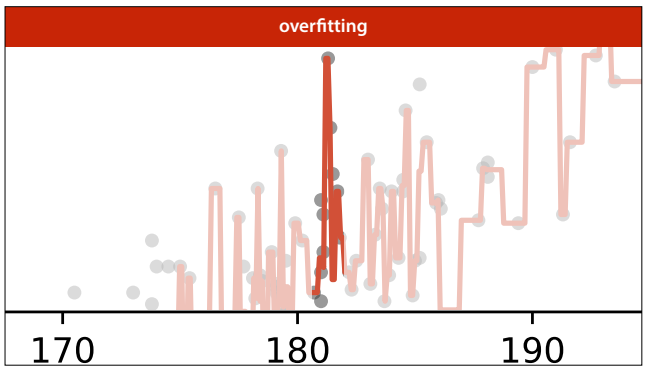
This is a difficult subject to explain precisely; however you look at it, it boils down to treating people with respect. What consists of respectful treatment, and how much of that people should be able to demand, is highly subjective. There are plenty of competitors to the Cloud Vision API that offer gender classification, so google is likely sacrificing some customers who are looking for the feature, and gaining others who value their stance.

If nothing else, this issue shows how far machine learning has come: our classifiers are no longer stumped by a judge's wig or a Beatle do, and they are certainly no longer confined to the laboratory. They are out there, making predictions on an international scale, so if we are the ones pushing them to production, we are the ones responsible for the consequences.

In this last video, we'll look a little deeper into the problem that is at the heart of machine learning. You may think the aim of fitting a machine learning model is to fit the training data as precisely as possible, but you would be mistaken. The aim is to find a model that **generalizes**.



To explain, consider our tree-based solutions to the classification and regression problems. Both cover most of the data perfectly. Every red and blue dot is perfectly classified and the regression line hits almost every dot in the dataset. Imagine you see a new person, and you are given their height. Would it make sense to use the model on the right to predict the length of their legs?

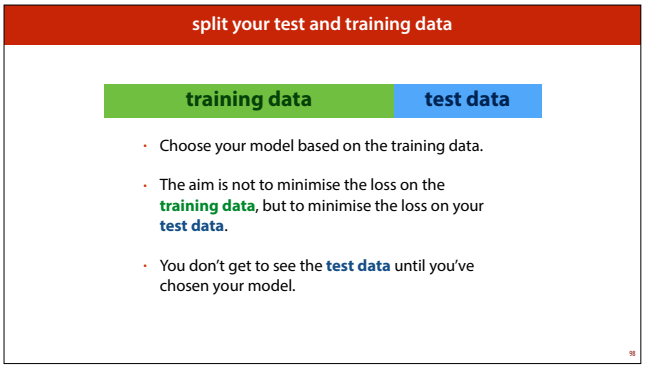


Let's look at this spike: the model seems to be convinced that people who are 183cm tall have proportionally much longer legs than people who are 182 or 184 cm tall. This isn't true of course, it just happens to be the case that in this area there was only one person in the data and they had long legs. The model is fitting details of the dataset that are *random noise*.

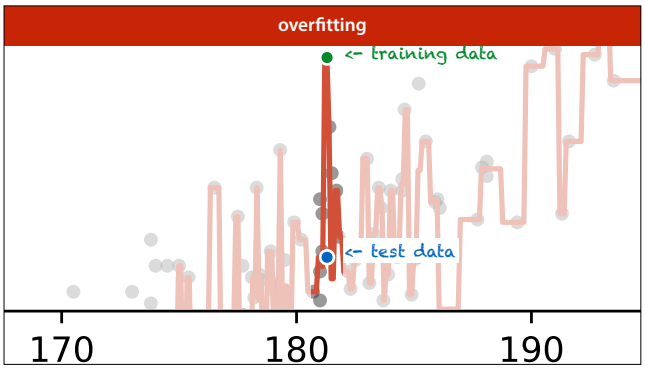
When a model overfits, we sometimes say that it is **memorizing** the data, when it should be **generalizing**.

This is the most important rule in machine learning. The regression tree had the lowest loss on the training data, **but it's actually the worst model** out of the three models we saw.

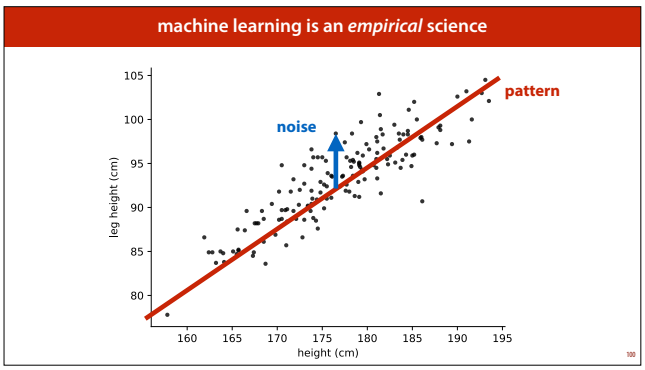
It means nothing how many of the *training* instances the model gets right. What we actually want is a model that does well on **new data;** data that it hasn't been trained on.



The simplest way to check this is to **withhold data**. You keep some data hidden from the model, and then check how well a particular model does on this part of the data. The data you show your model is called **training data**, the data you withhold is called **test data**.



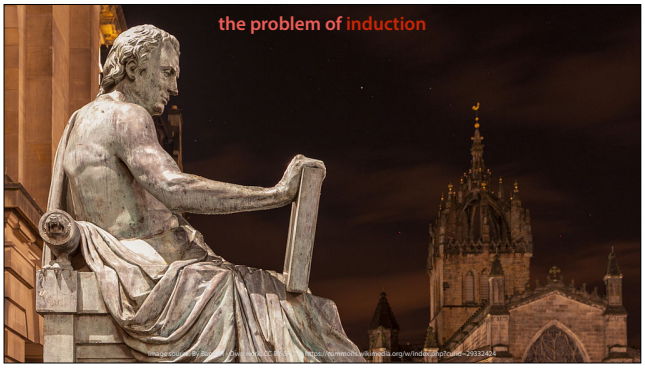
Now we see that the regression tree is a terrible model. The training data may show a spike at 183, but the test data will just follow the linear pattern.



The task is not to fit the training data as well as possible. It is to fit the **pattern** in the training data and discard the **noise**.

What is noise and what is pattern? There is no theoretical justification.

We can only try things and see if they work. Fundamentally, machine learning is an empirical science. Which is not to say that we don't use theory to build our models, and to help us guess what might work. But ultimately, the proof that something works is empirical, not theoretical.



The problem of how we learn (and how we can make machines that learn) is an instance of the *problem of induction*. It was first posed as a problem by 18th century philosopher David Hume (pictured).

Inductive reasoning is essentially a philosophical name for learning. We observe something happening a number of times, so we *infer* that it'll probably happen again the next time. We're not absolutely certain, and it doesn't follow logically, but we're sure enough to use that knowledge to our advantage.

This is very different from the *deductive reasoning* which philosophers had studied since antiquity. Deduction is rule-following. It's what computers do

best. In order to make computers do something like inductive reasoning, and in order to fully understand how *we* do it, we need to reduce it to rules. But Hume argued that inductive reason can not be proved to work by deductive methods.

---

deductive reasoning	**inductive reasoning**
All men are mortal Socrates is a man	The sun has risen in the east every day of my life
therefore Socrates is mortal	so it will do so again tomorrow
discrete unambiguous provable known rules	fuzzy ambiguous experimental unknown rules

102

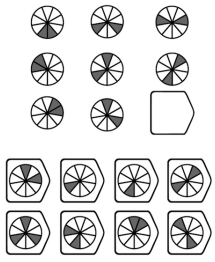
For deductive reasoning, we know the rules, and we understand them perfectly. For inductive reasoning the rules are not so clear. For instance, whenever I visit a funeral, I'm never the person being buried. Therefore, the more funerals I visit, the more certain I should be that next time it won't be my funeral. Clearly this is not the case (usually the opposite is true).

So, if inductive reasoning doesn't follow as a special case of deductive reasoning, and inductive reasoning applies sometimes and it doesn't at other times… how do we do it? Why is the funeral example obviously wrong, and the sun example obviously right? If inductive reasoning cannot be reduced to deductive reasoning, do we have any hope of reducing it to a computer program?

In many ways, the problem of induction is still unsolved. We can teach computers to learn pretty well these days, but we still don't fully understand what all the rules are.

---

finish the sequence



source: https://iqpro.org/

103

Machine learning is a bit like finish-the-sequence puzzles, that are often part of IQ tests.

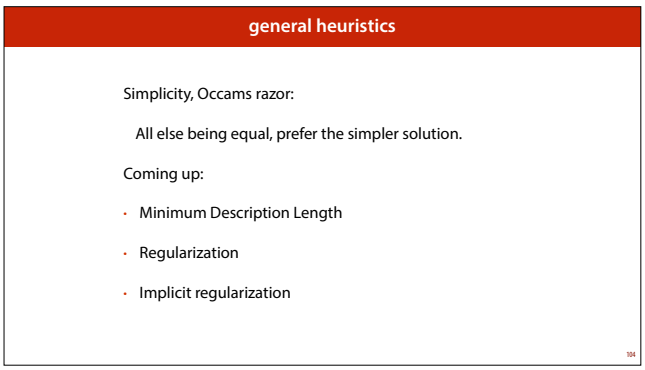
Some people get frustrated by puzzles like these, because the rules are not spelled out. We are supposed to infer the rules of this particular sequence and then apply those rules to find the missing element. Likewise, in machine learning, we are supposed to infer the pattern from the training set and apply it to the test set.

Obviously, the correct solution is the one that fits the test set as well, but we have to decide before we see the test set.

In this case, there are two ways to solve the puzzle. Reading from top to bottom each column contains a

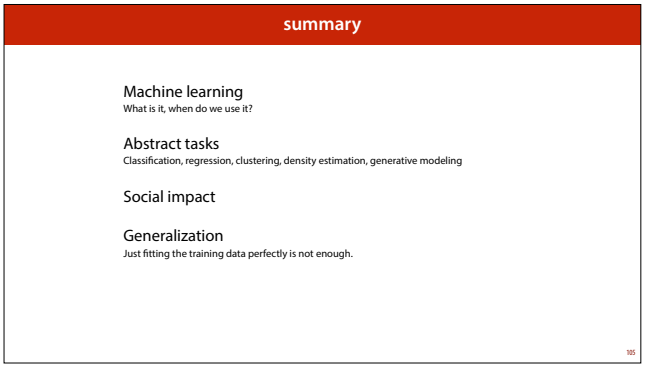
pattern that is rotated by three slices each step. Reading from left to right, the two slices in the pattern are pushed one slice further apart each step. In this case, both patterns lead to the same solution. But what if they didn't? Which solution are we supposed to prefer? What if we come up with a highly convoluted reason for preferring some other slice, why would that obviously be wrong?

The truth is, that while we have some heuristics for which solutions we tend to prefer, there is no general theory of learning that is always obviously correct. Ultimately, the solution to puzzles like these are appeals to *intuition*, and so is the solution to a machine learning problem. This is what makes machine learning so difficult, and what makes it so interesting.



That isn't to say we don't have a general idea of what makes one solution better than another. There are always exceptions, but in general, preferring simple solutions over complex ones seems to lead to good learning performance. That doesn't solve everything: we still have to make precise what simplicity *means* exactly, and we don't know, if the simple and complex solutions aren't equally good, how much simplicity we should sacrifice for a better solution.

In later lectures we'll look at some ways in which this intuition is made more precise in practice.



If you have any questions, please ask them on the Canvas discussion board if at all possible. If it's a personal question, or if you need to ask me personally for some other reason, please don't hesitate to email.